Assignment 3 Harris Corner Detector & Optical Flow

Daniel van de Pavert(11045418), Shuai Wang(13128051), Damiaan Reijnears (10804137)

September 2020

1 Harris Corner Detector

The *Harris Corner Detector* cleverly—but straightforwardly—makes use of *partial derivatives* of images for the detection of image features – a tool, for the purpose of edge detection, which we have thoroughly discussed in earlier lab reports. The algorithm is roughly based on the idea that partial derivatives of pixel intensity values, in the proximity of corner-like features in the original image, differ in both (x- and y-) directions (while differentiated intensity values in the vicinity of edges would typically only differ in *one direction* – the direction of the edge). Corners are particularly *distinctive* image features (and thus features of interest) as they are not *constant* (where edges often are). The mathematical foundation is based on a *sum of squared distances* between pixels, of which the *second moment matrix* (or a *structure tensor*) is a Taylor approximation whose function can intuitively be understood as a 'covariance matrix' for the gradient[1] as its *eigenvalues* measure the 'degree of change' (of pixel intensity values) in a particular direction. Thus, we could say that the gradient has 'more than one direction' in the vicinity of a corner. It may be clear that both eigenvalues of this matrix (as we, for now, assume images to be only two-dimensional, and thus two-dimensional structure tensors) are large if pixel intensity values change in both directions – which is an indicator of a corner. The determinant of such matrix, which, as it is roughly equivalent to the product of its eigenvalues, will then be a large number if pixel intensity values change in both directions, as, otherwise, the second eigenvalue would have caused the product of eigenvalues to result in a relatively small number. The determinant is used as it is less computationally expensive (Harris et al., 1988, p. 150). Since the gradient for a pixel is always based on its surroundings (the intensity values of all neighbouring pixels) the corner detection algorithm involves *shifting a window* across the image – if, for the pixel at the middle of the 'window,' the product of the eigenvalues of its second moment matrix is larger than a certain threshold, a corner is likely detected.

[1]The gradient (a vector with its components, and thus resulting direction, consisting of the partial derivatives, derived in each direction, of a point in the 'pixel plane') measures the direction and 'strength' of steepest ascend in pixel intensity values.

Resulting from the foregoing explanation, the Harris Corner Detector incorporates two parameters: a parameter which we define as n for the size of the $n \times n$ window and a parameter which we define as t for the threshold value. As images come (for example) in many different degrees of *brightness*, the ideal value for the threshold parameter t might vary per field of application. Thus, in our implementation, we have defined t in terms of a factor, f, of the maximum pixel intensity value for the whole image, so that $t = f \cdot max(I)$ (note that another feasible option would have been to *normalize* matrix $\mathbf{H}$). The effect of varying this parameter, f, on the number of returned corner coordinates is shown in figure 1. The results of running the algorithm are shown in figure 2 and 3.

The algorithm is rotation invariant as eigenvalues solely explain the *strength* of 'the degree of change' in a certain direction. No matter the direction, the corners always produce the same disruptions in two directions on the pixel point. However, the *eigenvectors* (directions in which the image 'changes most'), are not invariant. This phenomenon is illustrated in figure 4. For a rotation angle of 45°, the detector fails to detect one important corner point. This is partly due to the fact that the rotational images have a black background on the sides, which are detected as very evident corners, and causes the maximum encountered value in $\mathbf{H}$ to increase ($2.10 \cdot 10^9$ in figure 4b as opposed to $1.75 \cdot 10^9$ in figure 4e), on which the threshold is based. As the Gaussian filter is isotropic[2], it could not explain the oddity of figure 4b.

Although, in the original paper, Harris et al. (1988, p. 149) propose a cross-correlation operation with 'classic derivative kernels,' a *Sobel kernel* is often used, which is also an isotropic kernel (preserving rotation invariance) (Sobel and Feldman, 1968). We have compared both methods to obtain the Gradient's component images and, based on the results illustrated in figure 5, the 'classic derivative kernels' are preferred over the Sobel kernels, although the results indicate that Sobel kernels might work better on different types of images (mainly defined as having less *grain noise*).

Shi-Tomasi algorithm

The Harris Corner Detector first defines a *cornerness* value $H_{x,y}$ for every pixel in the input image. The calculation of this value is based on taking the determinant of a structure tensor (which roughly equivalent to the multiplication of its eigenvalues). A window is then shifted across the image to find points of interest. The Shi-Tomasi algorithm defines *cornerness* in *explicit* terms of the eigenvalues, instead of taking a less computationally expensive approach by calculating the determinant and *trace* (which, in that sense, makes the Harris

[2]MATLAB, *Apply Gaussian Smoothing Filters to Images*, `https://nl.mathworks.com/help/images/apply-gaussian-smoothing-filters-to-images.html`, accessed on September 24th, 2020

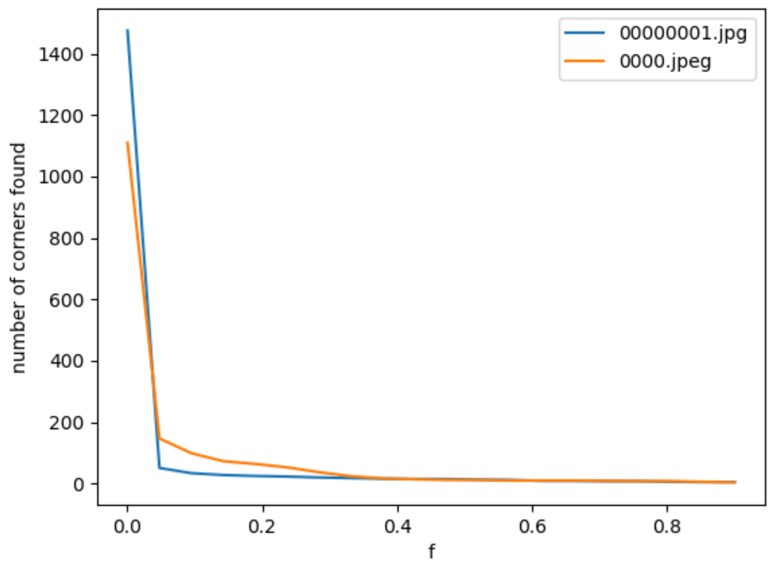
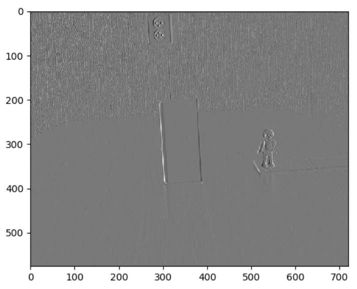


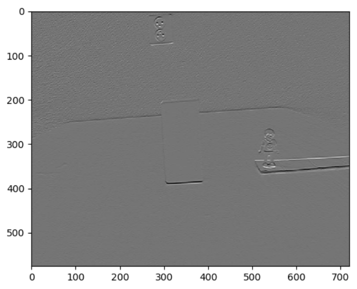
Figure 1: Number of returned corners for different values for f, while $n = 5$ and $\sigma = 1$

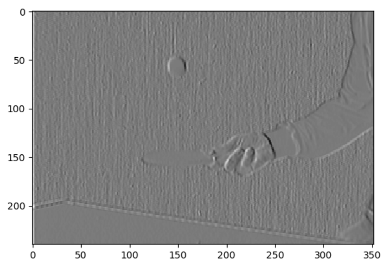
Corner Detector an *approximation* of the Shi-Tomasi algorithm). The definition of *cornerness* by Shi-Tomasi is shown in equation 1. If the lowest eigenvalue is big, both eigenvalues are big, and thus a corner is likely encountered. In the other cases (if a 'flat area' is encountered, both eigenvalues would be close to zero, while if an edge is encountered, just one eigenvalue would be large) equation 1 would return a low value.

One could easily infer analytically that this algorithm is rotation invariant (as eigenvalues simply refer to the 'degree of change,' rather than on direction, as explained in the previous subsection) and translation invariant (the same eigenvalues will simply 'turn op' elsewhere in the image after translation). However, the algorithm can not *scale invariant* as both the window size and the eigenvalues would change, since the partial derivatives are based on discrete pixel intensity value change over the course of one pixel. Scaling an image would either intensify or lessen the degree in which these pixel intensity values change *per pixel*.

$$H = min(\lambda_1, \lambda_2) \tag{1}$$


(a) Derivative in x-direction


(b) Derivative in y-direction


(c) Derivative in x-direction

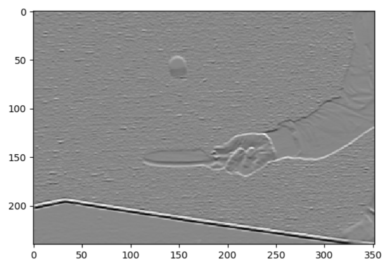
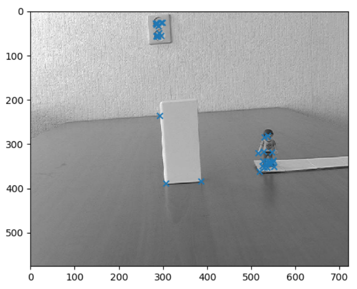

(d) Derivative in y-direction

Figure 2: Partial derivatives of *00000001.jpg* (a and b); and *000.jpeg* (c and d)


(a) *00000001.jpg*

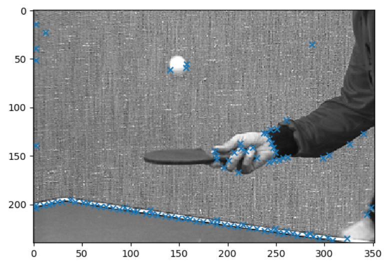
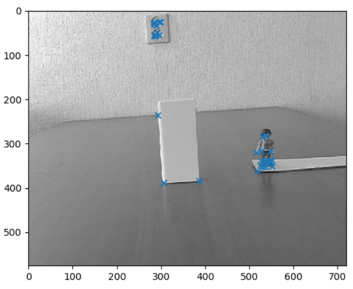
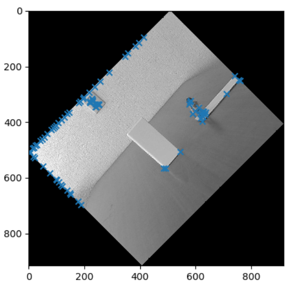

(b) *0000.jpeg*

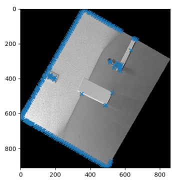
Figure 3: Harris Corner detection results with $\sigma = 1, n = 5, f = 0.1$



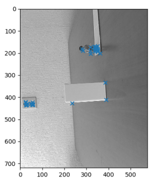
(a) Image rotation of 0°



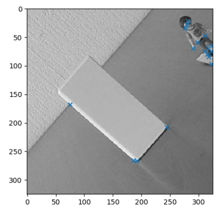
(b) Image rotation of 45°



(c) Image rotation of 60°

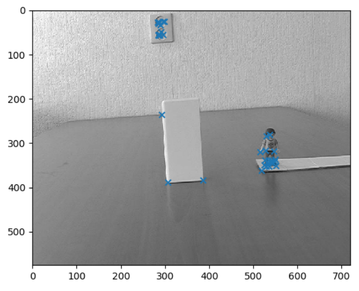


(d) Image rotation of 90°

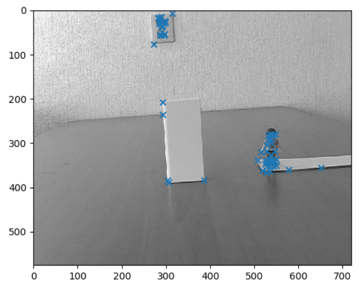


(e) Cut-out image rotation of 45° with $f = 0.075$

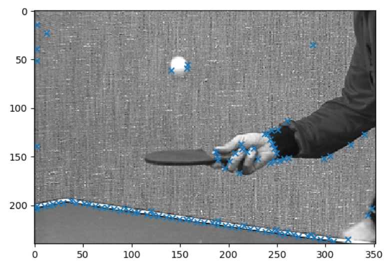
Figure 4: Corner detection for different angles with $n = 5, f = 0.1, \sigma = 1$



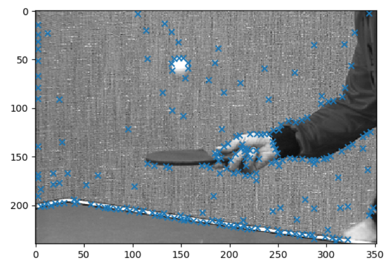
(a) Classic differentiation



(b) Differentiation using Sobel-kernel



(c) Classic differentiation



(d) Differentiation using Sobel-kernel

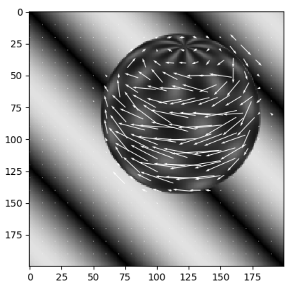
Figure 5: Corner detection results for different differentiation operators in *00000001.jpg* (a and b); and *0000.jpeg* (c and d), with $\sigma = 1, n = 5$ and $f = 0.1$.

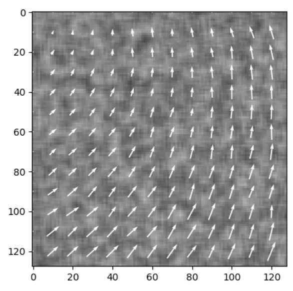
2 Optical flow

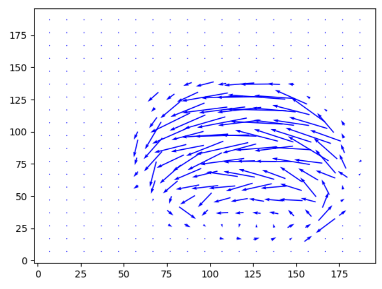
Optical flow techniques attempt to *estimate motion* from two (or multiple) *static* image frames. It is evident, as shown in the next section of this lab report, that features, such as corners, are of value for such kind of techniques as they can be *tracked* across different images taken at different times. It is therefore important that features are *distinctive*, such as is often the case for corners, so that they can be easily identified over several images (which, if we would have regarded points on edges to be distinctive features, would be more difficult, as points on an edge can be similar everywhere *along* that same edge). However, if one assumes *brightness constancy* (so that the intensity value of a pixel at a certain point would not change over time, even though it is now in a different location) and assumes motion to be *infinitely slow*, change can be measured by purely considering the partial derivatives in each dimension. The motion should then be equal to *zero*, as we assumed the motion to be *infinitely small.* Thus, essentially, this method assumes that the motion to be measured is slower than the change of the image derivatives itself. If we now assume images to have a third dimension of *time*, t, so that a pixel intensity value can be measured at $I(x, y, t)$, this constraint then translates to the 'main equation of optical flow,' as illustrated in equation 2. Part of this equation is equivalent to the dot product of the gradient of the intensity values of a point in the image, with a vector, $\boldsymbol{V}$, conveying the direction and strength of the optical flow (which we are trying to measure). Thus, the direction in which the image changes most rapidly, is projected onto the direction of motion. Note that this constraint holds for every pixel *independently*, which causes a problem: the equation is unsolvable for $\boldsymbol{V}$ as we have too much unknowns ($\boldsymbol{V}_x$ and $\boldsymbol{V}_y$) in only one equation (*one* gradient per pixel intensity value, obviously).

$$\frac{\partial I}{\partial x} \cdot \boldsymbol{V}_x + \frac{\partial I}{\partial y} \cdot \boldsymbol{V}_y + \frac{\partial I}{\partial t} = 0 = Grad(I) \cdot \boldsymbol{V} + \frac{\partial I}{\partial t} \tag{2}$$

The *Lucas-Kanade* algorithm and *Horn-Schunck* algorithm both handle the problem of 'having too less equations' to satisfy the above mentioned constraints on different ways. The Lucas-Kanade algorithm uses small windows (with which it filters over all values of the image), for which the constraint is assumed to hold *locally*, in which the system of equations is solved by using all pixel intensity values in that same window. In other words, the Lucas-Kanade algorithm assumes optical flow to be *constant* in a neighbourhood of pixel intensity values. The results of this algorithm are shown in figure 6. The Horn-Schunck algorithm introduces a second constraint – a constraint of 'smoothness of motion,' which essentially assumes motion *itself* to slowly change (so, not only the motion we initially try to measure, but also the change in that emotion itself), which is assumed to hold *globally.* Then, instead of trying to satisfy both constraints (the constraint of the amount of movement being actually zero, and having no change in movement at all), it tries to *minimize* the error with which these constraints are not satisfied, for every pixel in the image. Thus, while the *Lucas-Kanade* algorithm is an algorithm operating on *local scale*, the *Horn-Schunck* algorithm operates at *global scale*. No motion flow algorithms measure the flow of motion in flat areas (but may *assume* motion in flat areas instead), but rather the *apparent* motion image pixels. Even the human eye is unable to accurately detect motion in flat areas, as they do not incorporate any kind of texture (otherwise, it would not be a ‘flat area’). A comparison of both algorithms yields that the Lucas-Kanade algorithm is more robust and less prone to noise. An ideal approach would be to combine *local* and *global* methods (Patel and Shukla, 2013, p. 36).


(a) Measured optical motion flow of sphere image sequence


(b) Measured optical motion flow of synth image sequence


(c) Motion quiver plot of sphere image sequence

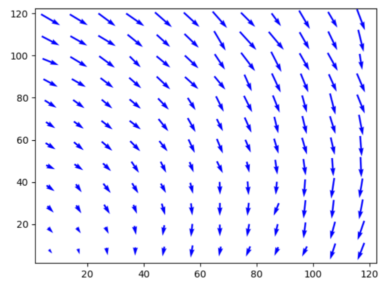

(d) Motion quiver plot of synth image sequence

Figure 6: Optical flow measurements using Lucas-Kanade algorithm

3 Feature tracking

disclaimer for tracking.py As it is quite a heavy process to track this way our computers were not fully up to the task and we mainly tested everything with the first few images of the pingpong image set. The two gif files are included in the assignment zip but the person toy gif might be a few frames short due to the processing speed is about 20 frames in 12 minutes on the used machine.

Question 2: If we perform feature tracking for objects in the video, we can globally track that object across frames, instead of re-sampling features for every frame. This, of course, again highlights the need for *highly distinctive* features.

References

Harris, C. G., Stephens, M., et al. (1988). A combined corner and edge detector. Citeseer.

Patel, E. and Shukla, D. (2013). Comparison of optical flow algorithms for speed determination of moving objects. *International Journal of Computer Applications*, 63(5).

Sobel, I. and Feldman, G. (1968). A 3x3 isotropic gradient operator for image processing. *a talk at the Stanford Artificial Project in*, pages 271–272.